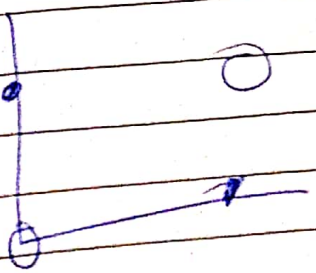
Bharath Kumar Thulasi
2017035

Neural Networks with only linear activations can never learn XOR, however much ever the hidden layers would.

Linear activation outputs $y = mx + c$
where x is input & y is output.
m is the weight & c is bias.

So suppose we have a neural network then to progress to the next layers we always have to do $y = mx + c$ i.e multiply input by a weight. The only way this could become nonlinear is if we had a term $(x \times x)$ i.e (input × input) which is clearly not possible in this case. And hence all the layers will keep passing a input × weight term, which stays linear. Hence this cannot be applied to XOR, which is a non-linear problem.



Cannot be linearly separable, we could need 4 inputs for that.

$f(x) = B(Ax + a) + b$
$= BAx + Ba + b$
$= (C_1 x + C_2)$ is again linear

4) different components of CNN are :-

i) convolution layer :-

It is used for feature detection/extraction mainly. CNN's use kernel to reduce the input features size (not in all cases), but mainly to capture important spatial features of the image input. It sometimes uses same kernel again & again for all inputs since it is not dense & is computationally inexpensive.

ii) Pooling ~~area~~ Layer :-

The main use of pooling layer i'e to use a kind of stride box, it to progressively reduces the spatial size of the input. This reduces no of inputs & parameters to the next layer which reduces the computations as well. It also downsamples the data.

~~Relu~~ Activation Layer

→ Activation functions are needed to make sense of non-linear inputs. Many activation layers like Sigmoid tanh & relu can be used for this purpose.

Dropout Layers

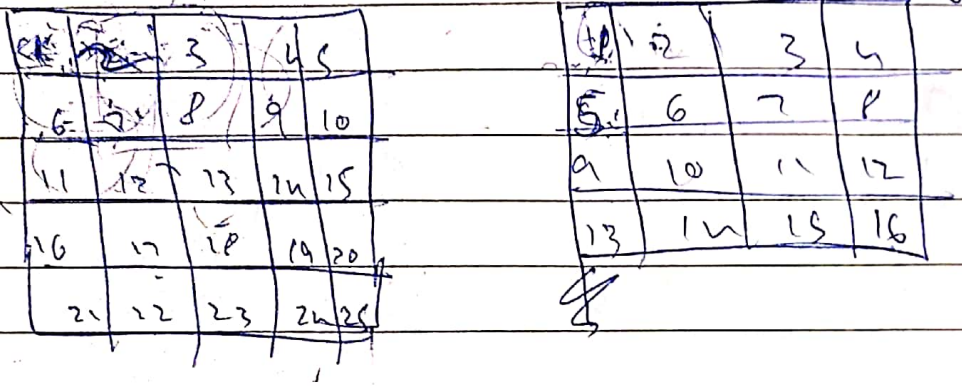
Dropout layers are used to drop some neurons randomly during the training. This is done to stop overfitting in some instances as the neuron might learn exact particular input values & over and increase variance.

Fully connected layers

Fully connected layers are used to do the final classification of a CNN. They are dense layers and output probabilities of the output classes for each input.

Q3) Assuming 2×2 kernel with stride 1 for first hidden layer.

So input 5×5 can be considered and next layer will have



So we have to make following connections

→ connect (1, 2, 1, 1)
" (1, 2, 2, 1)
" (1, 2, 6, 1)
" (1, 2, 7, 1)
- - - - - -

So similarly every neuron in 2nd layer will have 4 connections from previous layers

Also

Share (1, 2, 1, 1, 2, 2)
Share (1, 2, 2, 2, 3, 3)
Share (1, 2, 3, 3, 4, 4)
Share (1, 2, 4, 4, 6, 5)
. . . . .

Basically whenever the 2×2 kernel coincides with the inputs.

Code

Connect

count=0

→ for i in range(0,16) i + counts

connect(1, 2, i, i)

connect(1, 2, i+1, i + count)

connect(1, 2, i+5, i + count)

connect(1, 2, i+6, i + count)

if (i%5 == 0):

count += 1

→ increase
→ Modulas

increase
ie (1,4) + 1
= (2,1)

& add pairs

Share

iterate through 5x5 Matrix = M

→ a counter = (1,1)
b " = (1,1)
c " = (1,1)
d " = (1,1)

for i in rows
j in cols
if (i<4 && j<4) counters a+x4 + counter
add (M[i][j]) to hashMap[a]
increase a counters by 1
if (i<4 && j ≠ 1) counter b+x4 + counter b2
add (M[i][j]) to " [b]
increase b counter
if (i ≠ 1 && j ≤ 4) & counter c+x4 + counter c2
add M[i][j] to " [c]
increase c counter
if (i ≠ 1 & j ≠ 1) & counter d1+x4 + counter d2
add M[i][j] to " [d]
increase counter

hashMap
= [a → [ ]
[b → [ ]]
[c → [ ]]
[d → [ ]]

This will create a hash Map of shared connections b/w neurons for that weight.

for i in HashMap
    for j in range(len(i-1))
        ~~share(i,z~~
    for k in range(j, len(i))
        share(j, z, k[0], k[1], j[0], j[1])
                # to count pairs

Now connect 8×4 Matrix to 2×1 output

Matrix
2 rows

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

3 rows

1
2

connect
for i in 4
    for j in 4
        connect(2,3, i×4+j, 1)
        connect(2,3, i×4+j, 2)

No weights shared here in FC